CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Notice

- Hw5 posted, and will be due on this Sunday midnight.
- Hw1-Hw3 have been done grading

Discuss on Quiz

- The Binary Search
- The time complexity Problem
- for outer layer of the loop:
  - If it is linearly incremented or decremented (i.e. ++, +=2, --, -=2, …, O(N)
  - If it is times by 2 or divided by 2 per iterations: O(logN)
- For inner layer of the loop
  - if it is independent of the outer layer, the same policy applies.
  - If it is dependent of the outer layer, and both of the outer layer and inner layer are linearly incremented or decremented, then they both will be O(N)
  - If it is dependent of the outer layer, and one of them is times by 2 or divided by 2 per iterations, then more analysis is needed.
- Multiple the complexity order on each layer of the loop.

Discussion on HW

- Please do not change the part I provided
- Please add comments to each program:
  - Credit comment
  - General comment: the main purpose of the program
- RemoveDuplicate: a good representation of the Stack applications. The time complexity can be O(N), and the array implementation O(N^2)

Recap

- Evaluating postfix arithmetic expression
- The Priority Queue data structure
- Applications on Priority Queue
- Implementation of Priority Queue using Ordered Array.
- [Optional] convert infix to postfix

Learning Topics

- Efficiency of Stack, Queue and Priority Queue
- The linked list data structure:
  - Singly linked list
    - Single-ended singly linked list
    - Main operations on single-ended singly linked list and its time complexity
    - Applications on single-ended singly linked list
    - Double-ended singly linked list
    - Implementation of Stack and Queue using singly linked list
  - Doubly linked list

Efficiency of Stack, Queue and Priority Queue

Using the array implementation:

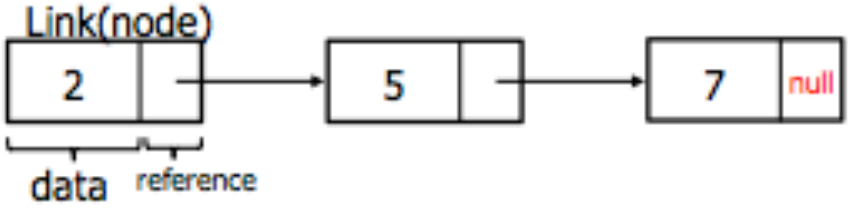
Operations	Stack	Queue	Priority Queue
push(enqueue)	O(1)	O(1)	O(n)
pop(dequeue)	O(1)	O(1)	O(1)
peek	O(1)	O(1)	O(1)
isEmpty	O(1)	O(1)	O(1)
isFull	O(1)	O(1)	O(1)
size	O(1)	O(1)	O(1)

Linked List-Introduction

- Array:
  - Main property:
    - Fixed length
    - Contiguous memory allocation
  - Advantages of array:
    - O(1) access to every element
    - Data locality in memory
  - Disadvantages of array:
    - Size is fixed
    - Some operations require shifting items

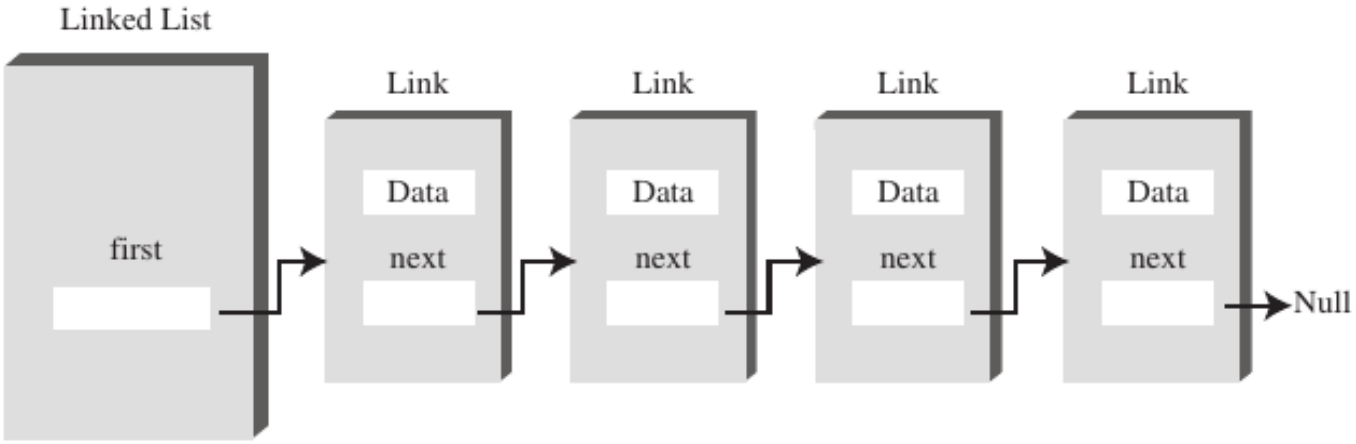
Linked List

- Linked list is a composed of a list of nodes (links) , each node (link) holds data and contains a reference to the next node (link)
- Main Property:
  - Dynamic memory allocation (resizable)
  - You can expand the list dynamically by just adding more nodes (links)
  - Do not Need to move elements when deleting or inserting
  - No data locality, as the memory is allocated dynamically.



Linked List

- It starts from *first link*(head), the last link points to null.



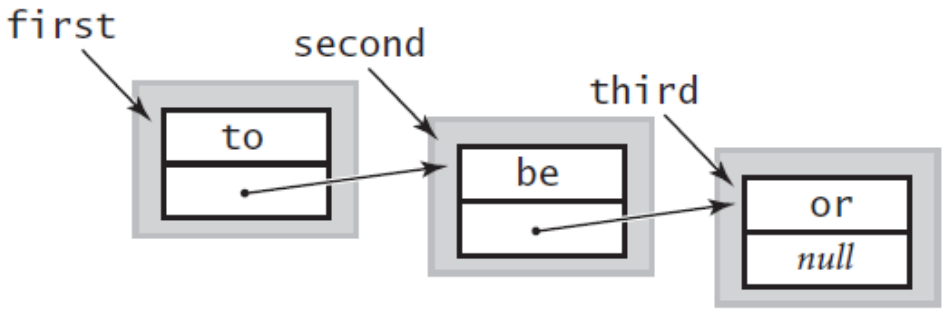
- **The class LinkedList** provides reference to the first link and implements methods(add, delete, …)

- **The class Link** contains data and reference to the next link.

Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 180

Single-ended Singly linked list

- Singly linked list: Each node (link) contains only one reference to its next node.
- Single-ended: the entire linked list is only accessible by the reference to its first node (head)



The Link Class

```
class Link
{
        public int data; // assuming integer data
        public Link next; // reference to the next Link
//-------------------------------------------------------------
        // Constructor
        public Link(int data)
        {
                this.data=data;
                next = null;
        }
}
```

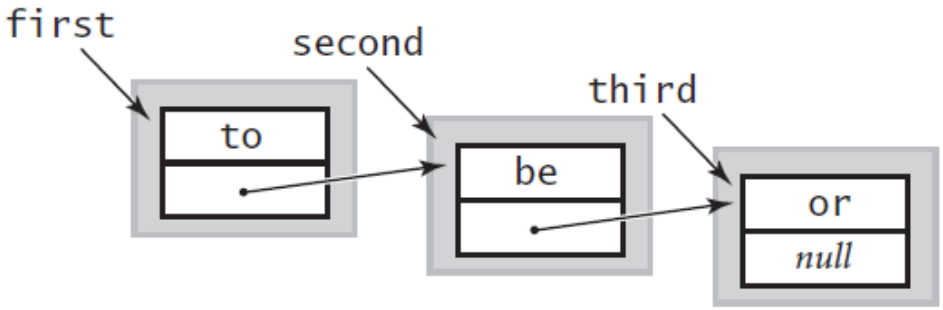
```
Link link1 =new Link (2);

Link link2 =new Link (5);

link1.next=link2;
```

The LinkedList Class

```
class LinkedList
{
        private Link first; // ref to the first link on list
// -----------------------------------------------------------
        public void LinkedList() // constructor
        {
                first = null; // no items on the list yet
        }
// -----------------------------------------------------------
        public boolean isEmpty()
        {
                return (first==null);
        }
        // other methods to insert, delete,…

        public void insertFirst(int data){}
        public Link deleteFirst() {}
        public void displayList() {}
}
```

Building a LinkedList

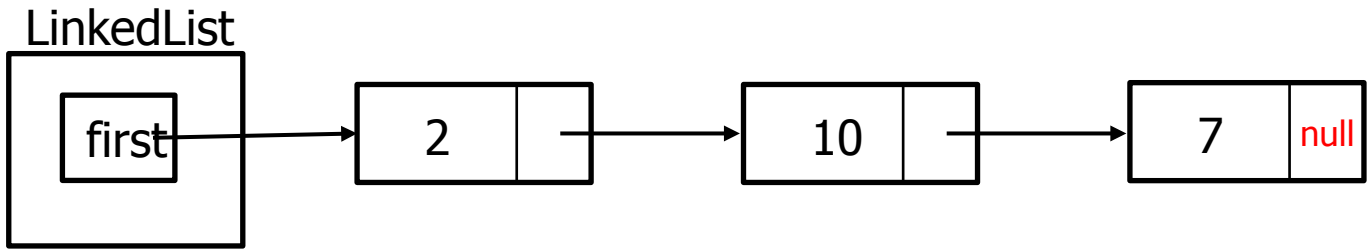
```
Node first = new Node();
first.item="to";

Node second = new Node();
second.item="be";
first.next=second;

Node third = new Node();
third.item="or";
second.next=third;
```



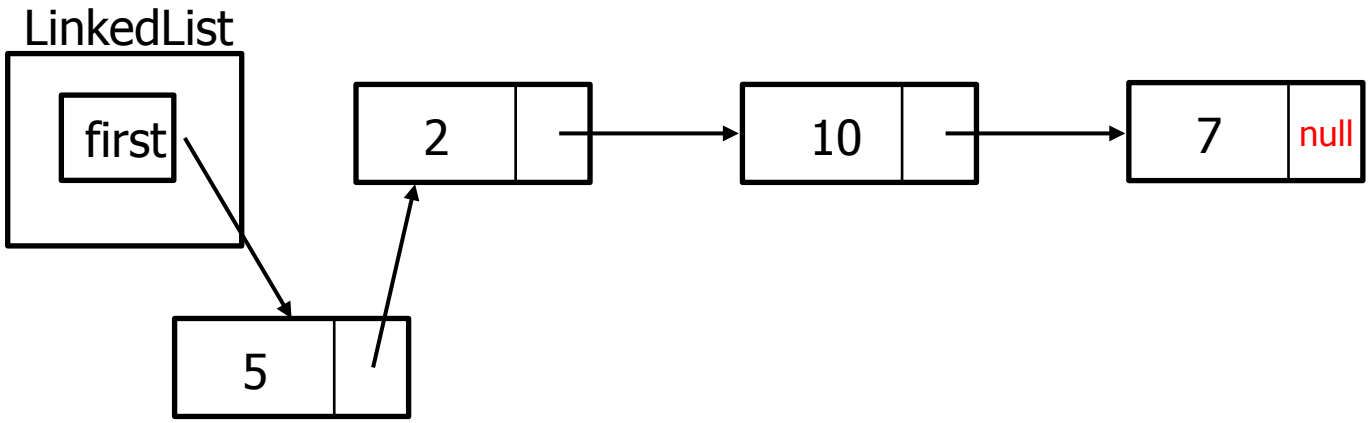
Single-end singly linked list: Operations

- Insert a new node at the beginning
- Insert a new node at the end
- Delete an existing node from the beginning
- Delete an existing node from the end
- Traverse the entire linked list
- Find a specific node in the linked list
- Delete a specific node from the linked list

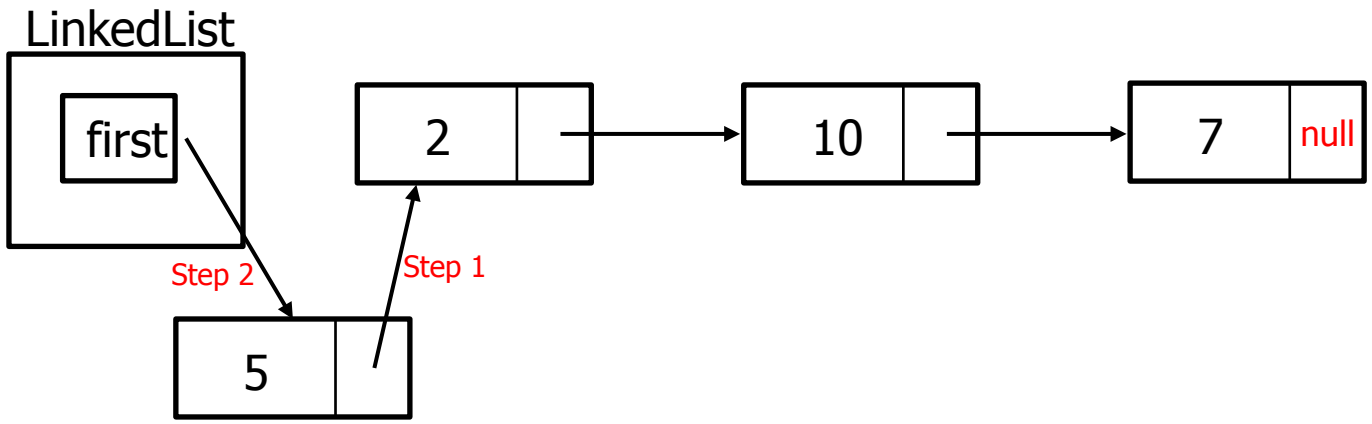
The insertFirst Method



insertFirst(5)

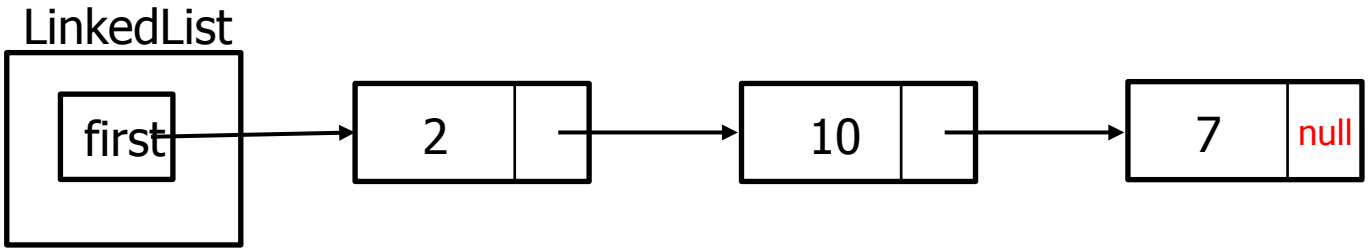


The insertFirst Method

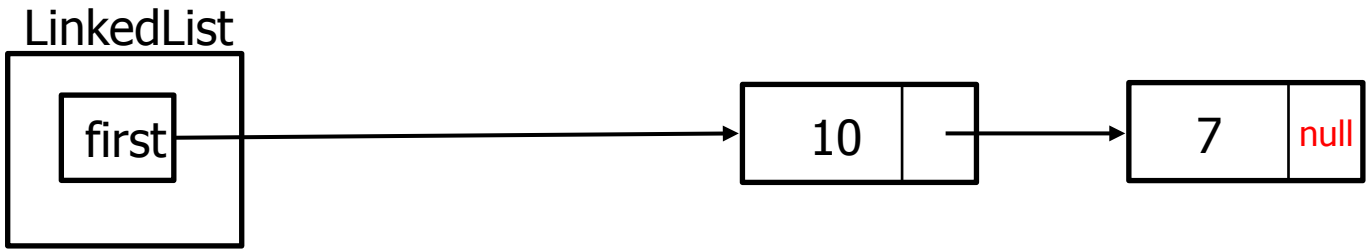



```
public void insertFirst(int data)
 {
        Link newLink = new Link(data); // create a new link
        newLink.next = first; // connect newLink to first
        first = newLink; // update first
}
```

The deleteFirst Method

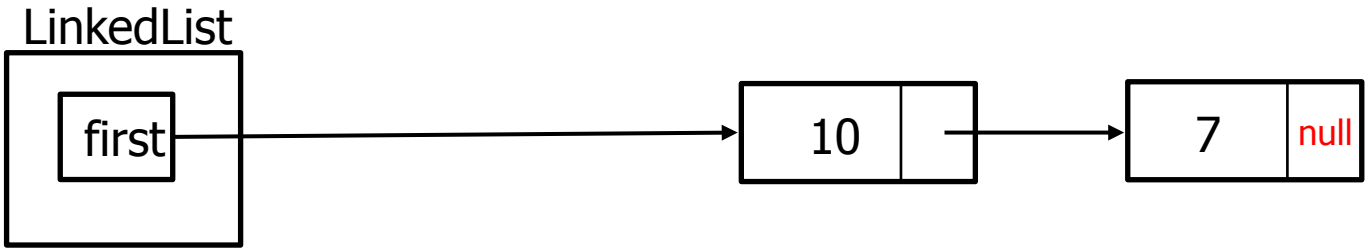


deleteFirst()



What happens to the deleted Link?

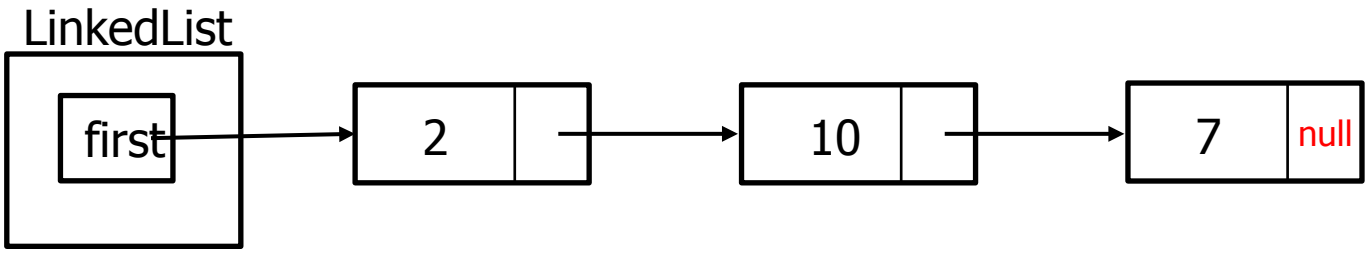
The deleteFirst Method




```
// delete from beginning and return the deleted first Link
public Link deleteFirst()
 {
            if (!isEmpty())
            {
                        Link temp = first; // save first to return link
                        first = first.next; // update first
                        return temp; // return deleted Link
            }
            else
                        return null;
}
```

Traverse the linked list: The displayList Method

- Start at the beginning of the list, Go over Links one by one using the next referer




```
public void displayList()
{
        Link current = first;
        while (current!=null) // until the end of the list
        {
                System.out.println(current.data);
                current = current.next;
        }
}
```

LinkedList Demo Class

```
class LLDemo
{
        public static void main(String[] args)
        {
                LinkedList myList = new LinkedList ();

                myList.insertFirst(4);
                myList.insertFirst (2);
                myList.insertFirst (3);

                myList.displayList();

                Link temp = myList.deleteFirst();
                System.out.println(temp.data);

                 myList.displayList();
        }
}
```

Finding and Deleting Specified Links

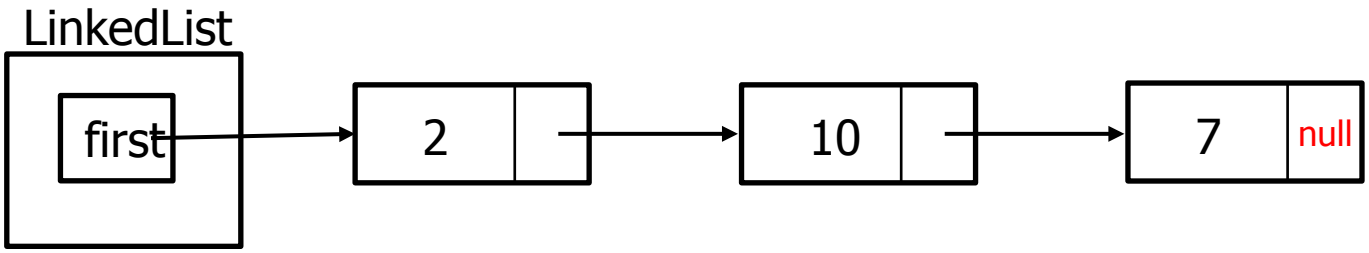
- Methods to search a linked list for a data item with a specified key value and to delete an item with a specified key value

```
public Link find(int key) // find link with given
key
{}


public Link delete(int key) // delete link with
given key
{}
```

- Need to traverse the List (like in displayList() method)

The find Method

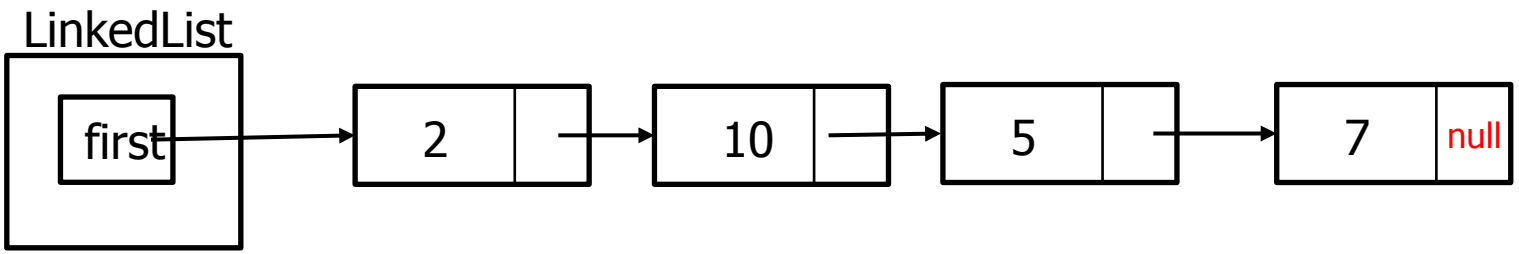
- Start at the beginning of the list, Go over Links one by one using the next reference.




```
public Link find (int key)
{
        if (isEmpty())
                return null;

        Link current = first;
        while (current!=null && current.data!=key)
        {
                current = current.next;
        }
        return current;
}
```

The delete Method

- Similar to find, but we need to keep the previous Link, as well.



delete(5);

The delete Method

```
public Link delete(int key)
{
        if (isEmpty())
                return null;

        Link current = first;
        Link previous = first;

        while (current!=null && current.data!=key)
        {
                previous = current;
                current = current.next;
        }
        if (current==first) //key was at the beginning
                first=current.next;
        else if (current!=null)
                previous.next= current.next;
        return current;
}
```